LYNXARM SIMULATION DOCUMENTATION

SHG Selten

Abstract

[Draw your reader in with an engaging abstract. It is typically a short summary of the document.
When you're ready to add your content, just click here and start typing.]

Sonny Selten (1654954)
Shg.selten@student.han.nl

Contents

Inleiding

Dit document dient als algemene beschrijving van de WOR-World lynxmotion simulatie oplevering. Het document bevat de klasse diagram, package diagram en node diagram samen met toelichten over hoe deze elementen samenhangen. Voor duidelijkere afbeeldingen omtrent de node graph en klasse diagram bekijk de UML folder.

Package Description

Lynxarm Simulation

Deze package is verantwoordelijk voor het simuleren van de Lynxmotion_ADL5 robotarm. Deze package bevat de launch file geschreven in python. Deze start de nodes op die nodig zijn voor het draaien van de simulatie en heeft een afhankelijkheid **CommandPublisher** package. De **CupSimulation** package is niet noodzakelijk om de robotarm te manipuleren.

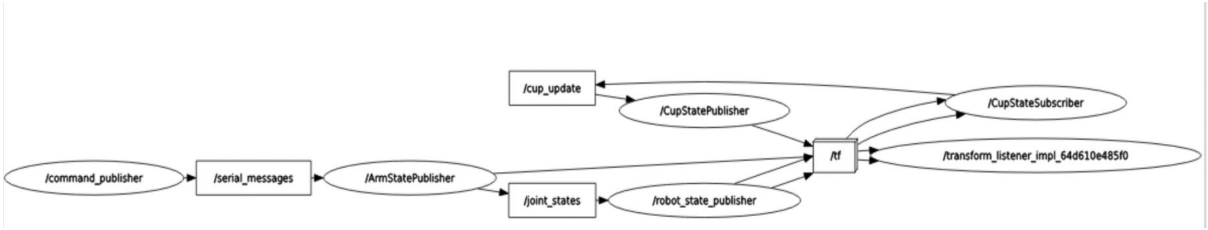
Cup Simulation

Deze package is verandwoordelijk voor het simularen van het cup object dat wordt vertegenwoordigd als een cilinder in de RVIZ simulatie en kan worden opgepakt door de gesimuleerde robotarm. Deze package bevat de Publisher en subscirber class die nodig zijn om het cup object op te pakken met de robotarm.

Command Publisher

Deze package is verantwoordelijk voor het versturen van seriele messages naar de **Lynxmotion Simulation** packge. Deze package bevat een demo class die het mogelijkmaakt voorgedefineerde messges te versturen die staan beschreven in de SerialMessages file.

Klasse & Node Description



Arm State Publisher

De **ArmStatePublisher** klasseis een C++ klasse bedoeld voor gebruik binnen een ROS 2 omgeving om de staat van een simulatie van een Lynxmotion_ADL5 robotarm te publiceren. Deze klasse erft van **rclcpp::Node**, wat aangeeft dat het een ROS 2 node is die communicatie binnen het ROS netwerk faciliteert. De kernfunctionaliteit van deze klasse omvat het publiceren van de staat van de robotarm met behulp van **sensor_msgs::msg::JointState** berichten. Dit berichtstype draagt gegevens over de positie van elk gewricht in de arm. De klasse maakt gebruik van timers om op regelmatige tijdstippen de armstaat te publiceren door middel van de **publishArmState** methode aan te roepen. Deze klasse ontvangt **std_msgs::msg::String** van de **CommandPublisher** package in het formaat van de standaart Lynxmotion_adl5 arm. Het PWM signaal wordt omgezet naar een radiaal en deze wordt gepubliceert in de RVIZ simulatie.

Cup State Publisher

De **CupStatePublisher** klasse functioneert binnen een ROS 2 systeem als een node voor het verzenden van seriële berichten Deze klasse erft van **rclcpp::Node**, wat aangeeft dat het een ROS 2 node is die communicatie binnen het ROS netwerk faciliteert. Het belangrijkste doel van deze klasse is het publisheren van de cup state. Voornamelijk voor het onderscheiden tussen de status (*IDLE, FALLING, GRABBED*). Deze class ontvangt **geometry_msgs::msg::TransformStamped** messages van de de **CupStateSubscriber** en verwerkt deze met de **HandleCup()** functie. Deze funcie verwerkt dan de status van de cup. Deze node is ook verantwoordelijk voor het publiceren van de cup URDF file over **/cup_description**.

Cup State Subscriber

De **CupStateSubscriber** klasse functioneert binnen een ROS 2 systeem als een node voor het verzenden van seriële berichten Deze klasse erft van **rclcpp::Node**, wat aangeeft dat het een ROS 2 node is die communicatie binnen het ROS netwerk faciliteert. Het belangrijkste doel van deze klasse is om te kijken on de cup zich binnen de **positiveMargin** en **negativeMargin** bevind door te luisten naar de **TF2_ROS::TransformListener** van de '*gripper_left*', '*gripper_right*' en '*hand*' die wordt opgeslagen in de **tf2_ros::Buffer** die de laatste 30 seconde bijhoud. Indien de bup binnen de marges is wordt er een **geometry_msgs::msg::TransformStamped** message verstuurd naar **CupStatePublisher** voor de nieuwe positie van de cup. Dit wordt gedaan op een 10ms interval van de timer.

Demo

De **Demo** klasse functioneert binnen een ROS 2 systeem als een node specifiek ontworpen voor het verzenden van seriële berichten Deze klasse erft van **rclcpp::Node**, wat aangeeft dat het een ROS 2 node is die communicatie binnen het ROS netwerk faciliteert.Het belangrijkste doel van deze klasse is het simuleren van de overdracht van seriële communicatie berichten naar de Lynxmotion_ADL5 robotarm. Bij de initialisatie van een **Demo** object, wordt een standaard pose ingesteld via de **initDefaultPose** methode. Het verzenden van seriële berichten met de **sendSerialMessage** methode gebeurt met een **std::string** bericht als parameter wordt doorgegeven hiermee kunenn seriele messages worden verstuur naar de **ArmStatePublisher**. Daarnaast biedt de **sendTestMessage** functie de mogelijkheid om de demo voor het oppakken van de beker en het laten vallen uit te voeren. Deze functie verstuurd voorgedefinieerde messages in uit **SerialMessages.hpp**. De functie **processSerialMessage()** checkt of er een custom string wordt meegegeven aaan de node. Als dit zo is wordt deze verzonden, anders wordt de demo sequentie uitgevoerd.

Message format

Het message format van de seriele messages die worden gestuurd naar de command publisher houden de originele formaat aan van de Lynxmotion_ADL5 arm met basis functionaliteit. Messagesbuiten dit format worden opgevangen door de error handler.

Format

Onderstaande tabel beschrijft het format van de seriele messages die kunnen worden verstuurd

Tabel 1

#	Definieert de servo die aangestuurd moet worden
P	Definieerd de puls breete van de servo
T	Definieerd die tijd die de servo er over moet doen om de positie te bereiken

Voorbeeld

Voorbeeld seriele message die kan worden verstuurd met beschrijving

Tabel 2

#3P2050T10000	Zet servo 3 op pwm 2050 binnen 10000 ms

MoSCoW

PA01	Alle code is gepackaged volgens de ROS-directorystructuur.	SHOULD	
PA02	Package is te bouwen met colcon op ROS2 Humble Hawksbill	MUST	
PA03	De applicatie wordt gebouwd met C++ volgens de Object Oriented principes die je geleerd hebt bij eerdere courses.	MUST	
PA04	Alle code voldoet aan de ROS C++ Style Guide.	SHOUD	

VS01	De virtuele controller luistert naar een topic waarop string messages in het formaat van de SSC-32U 1 worden geplaatst. Van de interface moeten ten minste commando's zijn opgenomen voor het verplaatsen van de servo's met een ingestelde duur en het stoppen van de servo's.	MUST	
VS02	De virtuele controller reageert op het topic (zie eis VS01) door bijbehorende joint_state messages te publiceren.	MUST	
VS03	De virtuele robotarm wordt gevisualiseerd in Rviz (een URDF-model van de arm is beschikbaar op OnderwijsOnline).	MUST	
VS04	De virtuele robotarm gedraagt zich realistisch m.b.t. tijdgedrag (servo's roteren kost tijd en gaat geleidelijk).	MUST	
VS05	De virtuele robotarm kan op een willekeurige plaats in de virtuele wereld geplaatst worden.	SHOULD	

VC01	Er kan op een willekeurige plek in de virtuele wereld een bekertje geplaatst worden.	SHOULD	
VC02	Publiceert een 3D-visualisatie van het bekertje voor Rviz.	MUST	
VC03	Detecteert de relevante punten van de gripper.	SHOULD	
VC04	Visualiseert de gedetecteerde punten van de gripper.	COULD	
VC05	Visualiseert wanneer de gripper het bekertje vastheeft.	SHOULD	
VC06	Het bekertje beweegt mee met de gripper (als hij vastgehouden wordt).	MUST	
VC07	Bekertje is onderhevig aan zwaartekracht wanneer losgelaten.	MUST	
VC08	Bekertje bepaalt en publiceert zijn positie.	MUST	
VC09	Bekertje bepaalt en publiceert zijn snelheid.	SHOULD	
VC10	Snelheid wordt getoond met rqt_plot.	COULD	

DI01	Een demoscript stuurt over de tijd een sequentie van commando's naar de armcontroller	MUST	
DI02	Locatie van het bekertje wordt in de roslaunch-configuratie bepaald.	COULD	
DI03	Locatie van de arm in de wereld wordt in de roslaunch-configuratie bepaald	COULD	

DM01	Beschrijft hoe de code gebouwd kan worden.	MUST	
DM02	Beschrijft stap voor stap hoe de arm bewogen kan worden middels enkele voorbeelden	MUST	
DM03	Beschrijft welke eisen gerealiseerd zijn. En geeft hierbij een (korte) toelichting.	MUST	

DD01	Beschrijft de structuur van de package (Nodes, topics, messages, et cetera).	MUST	
DD02	Beschrijft de structuur en samenhang van de broncode (class-diagrams, beschrijving, et cetera).	MUST	
DD03	Beschrijft hoe het gedrag van alle belangrijke componenten gerealiseerd is.	COULD	
DD04	Beschrijft de API van alle publieke interfaces.	SHOULD	

Bijlage

Class Diagram

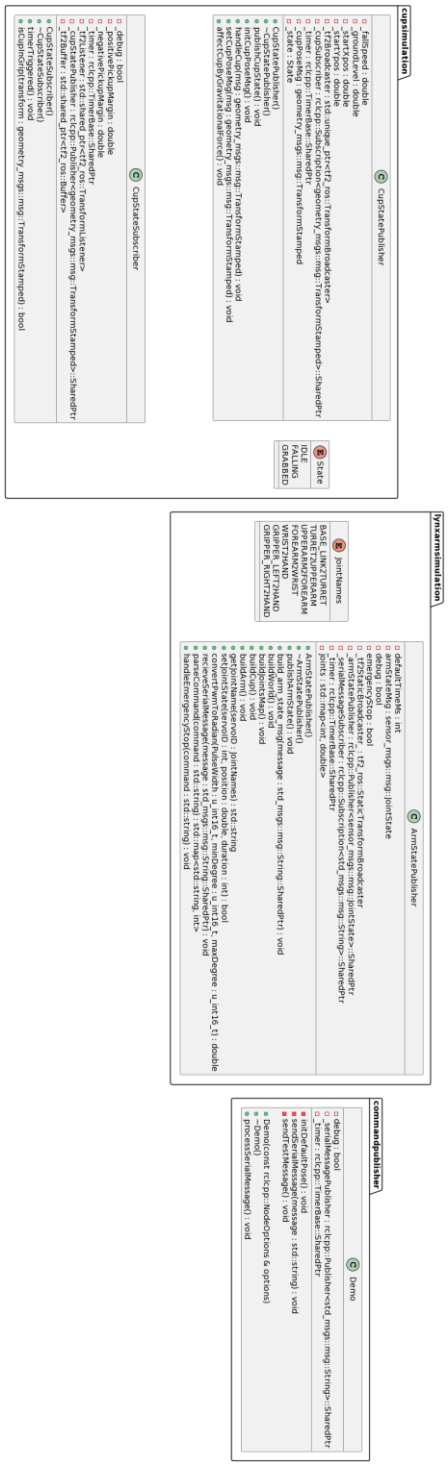


Figure 1 class diagram

Package Diagram

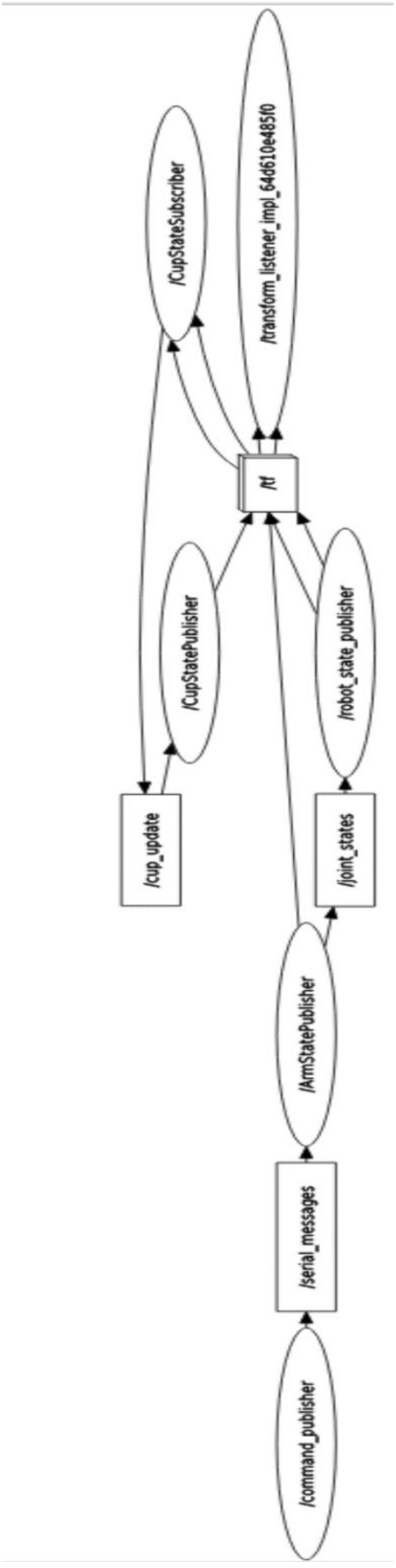


Figure 2 node graph